Design of Floating Point Arithmetic Logic Unit with Universal Gate

Shraddha N. Zanjat
Electronics (Communication)
S. D. College of Engineering
Wardha, India

Dr.S.D.Chede
Electronics and Telecommunication Engineering
Om College of Engineering
Wardha, India

Prof.B.J.Chilke
Electronics (Communication)
S. D. College of Engineering
Wardha, India

***Abstract*— A floating point arithmetic and logic unit design using pipelining is proposed. By using pipeline with ALU design, ALU provides a high performance. With pipelining plus parallel processing concept ALU execute multiple instructions simultaneously. Floating point ALU unit is formed by combination of arithmetic modules (addition, subtraction, multiplication, division), Universal gate module. Each module is divided into sub-module. Bits selection determines which operation takes place at a particular time. In this design of universal gate perform logical operation such as AND,OR,NOT,NOR,NAND operation also in this work area and delay parameter are reduces. The design is and validated using vhdl simulation in the xilinx13.1i software.**

Keywords- ALU - Arithmetic Logic Unit; Floating Point; Top-Down Design; Test-Vector; Validation.

I. INTRODUCTION

Floating point describes a system for representing numbers that would be too large or too small be represented as integers. Floating point representation is able to retain its resolution and accuracy compared to fixed point representation. IEEE specified standard for floating-point representation known as IEEE 754 in 1985.

The IEEE 754 floating point format consists of three fields. **Sign bit:** 1 bit .It is 1 for a negative number and 0 for positive number.

Exponent: 8 bits. The exponent represents a power of two.

Mantissa: Final portion of word (23 bits) is the significant that is also called as mantissa. Mantissa is a Fractional part.

Arithmetic logical unit is a combinational network that performs arithmetic and logical operation on the data. For Computation input data is given to A.L.U. code is also given from control unit, according to that code it compute the result. The ALU with floating point operations is called a FPU. Pipelining plus parallel processing execute is used to execute multiple instructions simultaneously. The cycle time of the processor is reduce .If pipelining is use, the CPU Arithmetic logic Unit can be design faster. It increases the overall performance of a system.

In the floating point ALU with universal logic gate we can perform addition, subtraction, multiplication, division operation and logical operation with less delay and less area.

II. PROBLEM DEFINITION

An A.L.U. performed addition, subtraction, multiplication, and division operation, latter on this, they add the logical operation which include AND, OR, NOT gate but the hardware complexity in terms of synthesis, delay and area is also more with less accuracy. So to avoid this problem we are designing 32 bit floating point A.L.U. which perform arithmetic operation which include addition, subtraction, multiplication, division operation and design of universal logic gate with high accuracy, high speed ,less delay and less area.

III. LITERATURE SURVEY

From the overall analysis we can say that floating point ALU was design that performed arithmetic operation which include addition, subtraction, Multiplications, and Division operations. That ALU was designed using pipelining because of that that the speed of ALU is increases and it executes multiple instructions simultaneously. That ALU was designed with Top down approach .In that ALU problem is that a more delay, more area , and hardware complexity in terms of synthesis is also more. This conclusion is made in [11], [4] and [2]. After that one ALU was designed that performed Addition, subtraction, multiplication, division, AND, OR, NOT operation [6].

Same designing method was used .Same problem is present in this ALU also. Our approach is that to design 32 bit floating point ALU with universal logic gate, which can perform Addition, subtraction, multiplication ,division , Logical operation with less delay and less area.

IV. DESIGN METHODOLOGY

Design Method includes the following modules:

1) Designing of multiplication module.

2) Designing of addition/subtraction module.

3) Designing of Division module.

4) Designing of universal gate module.

5) To study the concept of pipelining, Parallel processing which is used to execute multiple instructions simultaneously

6) Comparison and study of the results

V. ROBABLE OUTCOME

Proposed work will result 32 bit floating point A.L.U. with universal logic gate, which perform addition, subtraction, multiplication, division and designing of universal logic gate. This will meet the following specifications:

- Reduce Area
- Less delay

VI. IMPLEMENTED WORK

A. *Pipelined Floating Point Multiplication Module*

In this 32 bit pipelined floating point A.L.U. is designed with four stage pipelining. Which has two 32 bit numbers given as input and got result is a 32 bit number, which is a multiplication of that two 32 bit numbers .

In floating point numbers, multiplication of two numbers is basically performed through adding their exponents and multiplying their mantissas. Multiplier Module: Floating point multiplier unit block diagram is illustrated in Fig. 1

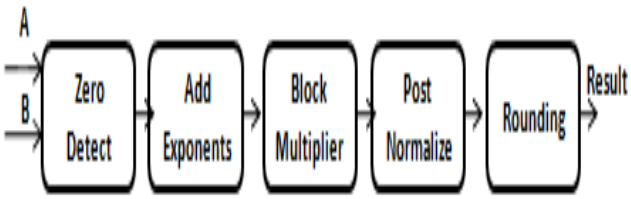

Figure: 1 Block Diagram of Multiplier

1) Zero Detect: It is used to set a zero flag when any of the input operands is zero. This avoids unnecessary calculations throughout the multiplier module when a zero input is applied. The sign, exponent and mantissa of each input operand are separated to be manipulated differently throughout the multiplier module. To prepare the mantissas for the multiplication operation, the dropped implied bit of each mantissa is set to "1".

2) Add Exponents: To determine the resultant exponent, the exponents are added and a 127 bias is subtracted. The bias subtraction compensates for the bias having been added in both exponents. The result is fit into a 10 bit exponent to allow checking for overflow or underflow in the post multiply normalization step. Since the 8 bit exponent of a single precision floating point number is an unsigned number, the 10th bit set to '1' indicates an underflow. An overflow is detected when the 10th bit is '0' and the 9th bit is '1'. Fig.1. Block Diagram of Floating Point Multiplier Module

3) Block Multiplier: Multiplies the two 24 bit mantissa of operands A and B. the bottle neck of the design is the 24*24 bit multiplier used to calculate the resulting 48 bit mantissa. To increase the maximum operating speed of the multiplier, the proposed design breaks up the 24*24 bit multiplication of operands A and B into nine 8*8 bit multiplications where each mantissa is sliced into three 8-bit slices such that A=A2A1A0 B=B2B1B0. Then, B0 is multiplied in A2, A1 and A0. Each of these three 8*8 bit multiplications gives a 16 bit result. The three 16 bit results are properly manipulated to give a 32 bit result Ro of the 24*8 bits multiplication operation (i.e. A*B0). In a similar manner B1 and B2 are multiplied in A2, A1 and A0 to give R1 and R2. R1 and R2 are properly shifted to be added to R0 thus giving the 48 bit result mantissa. The result sign is determined through a simple XOR operation.

4) Post Normalize: The implied bit is located and dropped. The 48 bit mantissa from the multiplication operation is then truncated to a 26 bit mantissa which is the 23 bits assigned for the mantissa along with three extra bits to increase the accuracy of the rounding process. These three extra bits are the guard, round and sticky bits. The guard

nd round bits are just an extension added to the mantissa to allow for extra precision. Sticky bit is the logical "Or"ing of all the truncated bits. The exponent is adjusted and checked for overflow or underflow and the appropriate flag is set accordingly.

5) Rounding: The 26 bit resultant mantissa is rounded to a 23 bits using the REN technique. After rounding, the exponent is checked again for possible overflow. Finally, the sign, exponent and mantissa are appended together to give the single precision floating point multiplication result in the IEEE format along with the overflow and underflow flags.

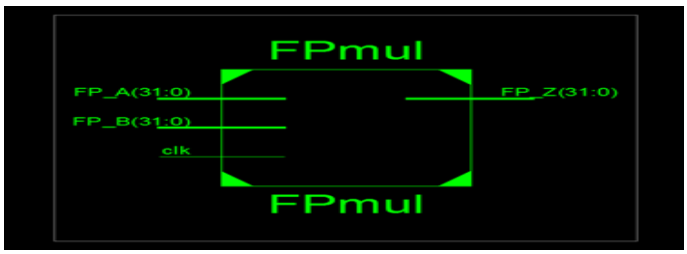

Figure 2: RTL Schematic of Multiplier

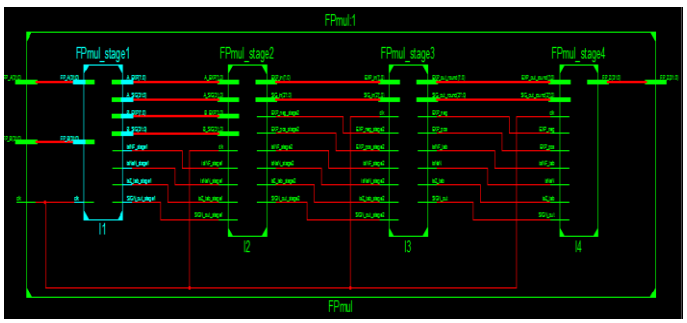

Figure 3: RTL Schematic of Multiplier

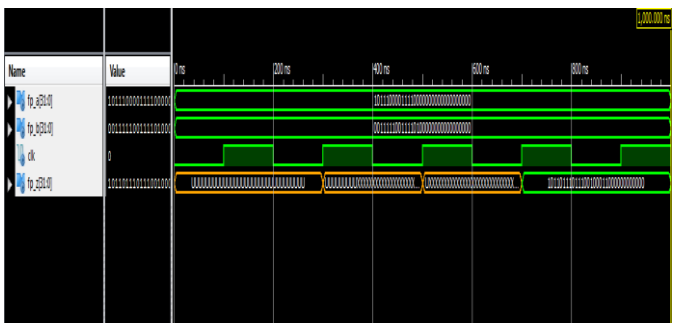

Figure 4: Simulated Result of Multiplier

B. Adder/Substractor Module

In floating point addition (or subtraction), the exponents of the two floating point operands must be equalize. So in the adder/subtracter unit, the exponent of the smaller number is incremented such that both exponents are equal and the mantissa of the small number is then shifted right 'n' times where 'n' is the difference between the large and small exponents. After the addition/subtraction operation is performed, the resultant mantissa is normalized using the LOD method. The LOD detects the most significant '1' by counting the number of zeros (nz) before the most significant '1'. The mantissa is then shifted left 'nz' times. Due to the excess shifting required before and after the addition/subtraction operation in the pre-normalization of the smaller number and the post normalization of the resultant mantissa respectively. To increase the maximum operating speed of the adder/subtracter unit all the shift operations in the pre-normalization and post normalization steps were performed through barrel shifting. Barrel shifting has the advantage of shifting the data by any number of bits in one operation which makes barrel shifting suitable for the shifting operations required in the adder/subtracter unit that can be a shift by any number between 1 and 253 depending on the difference between the exponents of the input operands. LOD Floating Point Adder/Subtracter Unit. The general block diagram of the floating point adder/subtracter module is illustrated in Fig. 5.

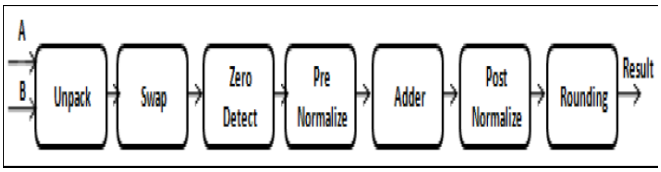

Figure 5: Block Diagram of Adder_Substractor

1) Unpack: The sign, exponent and mantissa of both operands are separated. A flag, a equal b flag, is set if both inputs are equal. The a equal b flag will be used if the effective operation, determined in the adder/subtractor module, was subtraction to set a flag indicating a zero output. This prevents unnecessary addition/subtraction and pre normalization operations from taking place.

2) Swap: Inputs are swapped if necessary such that operand A carries the larger floating point number and operand B is the smaller operand to be pre- normalized. A swap flag is set if the operands were swapped to be used in determining the effective operation in the adder/subtracted module.

3) Zero Detect: An appropriate flag is set if one or both input operands is a zero. This helps avoid unnecessary calculations and normalizations when a zero operand is detected. The resultant exponent and the difference between the two exponents are determined here.

4) Pre-normalize: The smaller mantissa, of operand B, is pre-normalized, that is it's shifted by the difference between the two input exponents. Three extra bits the guard bit, the round bit, and the sticky bit are added to both mantissas to increase the accuracy of the performed operation (addition or subtraction) and to be used in the rounding process. Sticky bit is the logical "Or"ing of any bits that are dropped during the pre-normalization of operand B.

5) Adder/Substractor: The effective operation to be performed is calculated according to the signs of operands A & B, the input operation and the swap flag. The effective operation is performed and the zero flag is updated if the effective operation is subtraction and the aequalb flag is set.

6) Post Normalize: The resultant mantissa is normalized after the leading one is detected using the LOD method. The resultant exponent is adjusted accordingly.

7) Rounding: The resultant mantissa is rounded using the REN technique. The final output is given in IEEE format.

In the 32 bit floating point ADDER/SUBSTRACTOR we are giving two 32 bit numbers, and clock signal is also given for synchronization purpose. ADD_SUB signal is given when this signal is '1' it perform addition operation, When ADD_SUB signal is '0' it perform subtraction.

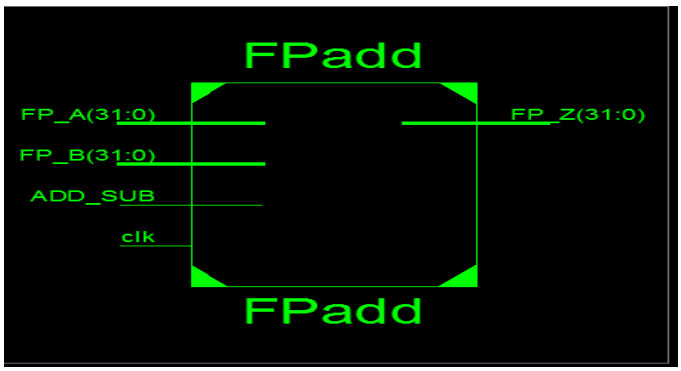

Figure 6: RTL Schematic of Adder_Substractor

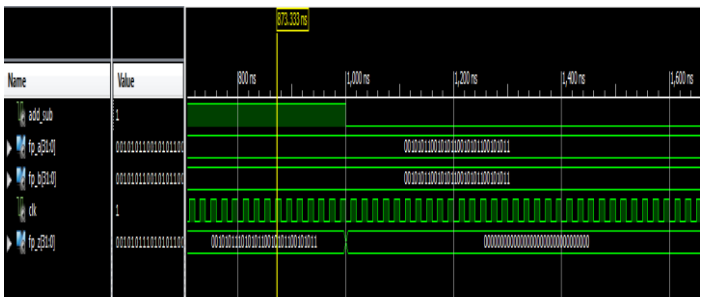

Figure 7: Simulated Result of Adder_Substractor

C. Division Module

Process of multiplication and division is almost same only difference lies is, for division exponent of two operand are subtracted and their significant are divided. If the result lies outside the permitted range normalization is required.

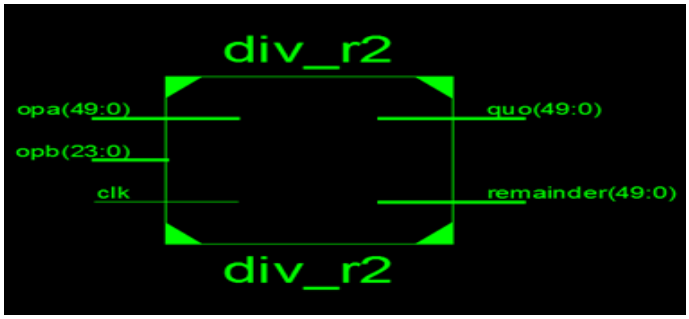

Figure 8: RTL Schematic of Division module

D. Universal Gate Module

By using universal gate that is NAND gate we can perform logical operation such as NOT, AND, NAND, OR, NOR operation.

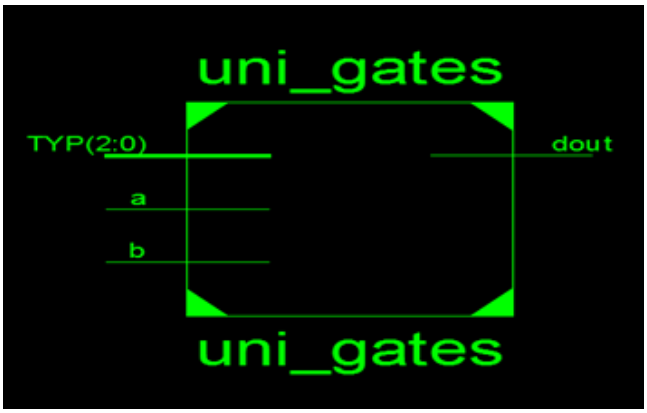

Figure 9: RTL Schematic of Universal Gate module

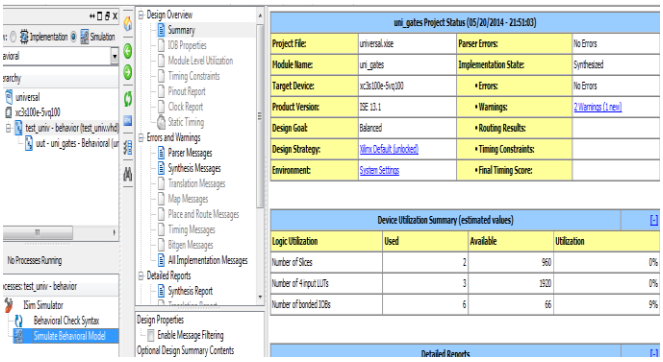

Figure 10: Area Representation of universal Gate Module

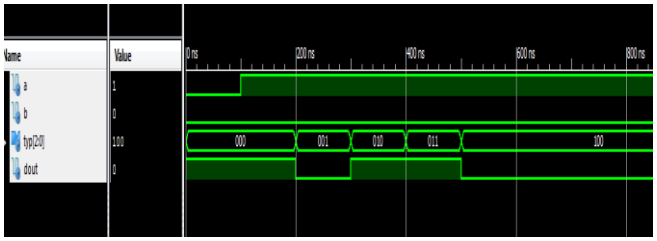

Figure 11: Simulated Result of Universal Gate Module

VII. FLOATING POINT UNIT

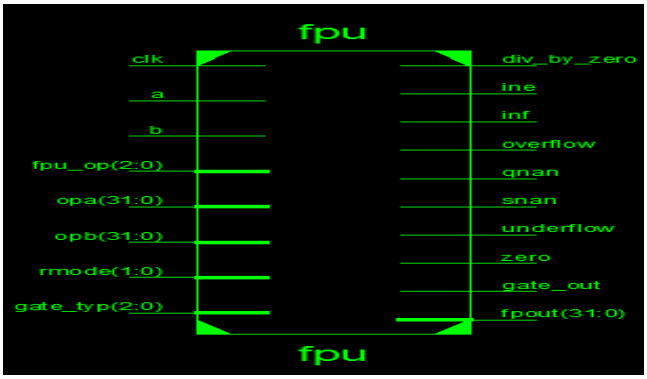

Figure 12: RTL Schematic of FPU

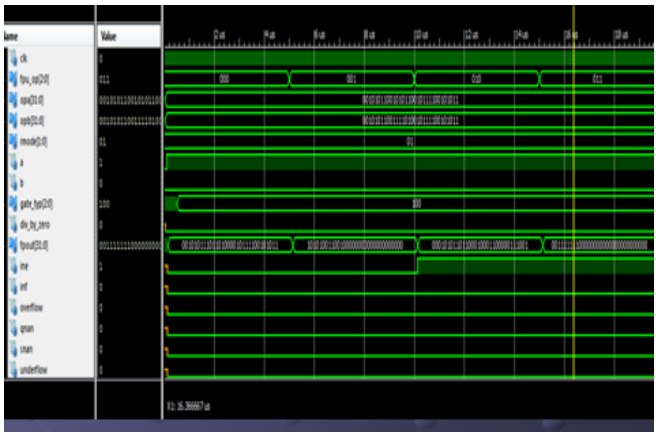

Figure 13: RTL Schematic of Universal Gate module

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	434	53,248	1%
Number of 4 input LUTs	3,517	53,248	6%
Number of occupied Slices	1,930	26,624	7%
Number of Slices containing only related logic	1,930	1,930	100%
Number of Slices containing unrelated logic	0	1,930	0%
Total Number of 4 input LUTs	3,522	53,248	6%
Number used as logic	3,507		
Number used as a route-thru	5		
Number used as Shift registers	10		
Number of bonded IOBs	116	448	25%
IOB Latches	1		
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number of DSP48s	4	64	6%
Average Fanout of Non-Clock Nets	3.64		

Figure 13: Area Representation of FPU

VIII. COMPARISON OF RESULT OF FPU

TABLE I. COMPARISON

Logic Utilization	Utilization	Utilization (Implemented ALU)
Number of Slices	360%	7%
No Of slice flip flip-flop	70%	1%
No of 4 input LUTs	336%	6%
Number of bonded IOBS	96%	25%
Delay	77.941ns	32.487nsns

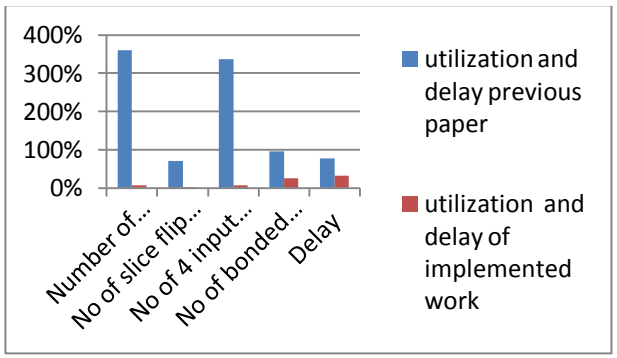

Figure 14: Graphical Representation of Delay and Area

ACKNOWLEDGMENT

I would like to thank my guide Dr. S. D. Chede and Co guide Prof. B.J.Chilke,for their guidance and support and Department of Electronics Engineering (Comm.), S. D. College of Engineering, Wardha.

REFERENCES

[1] Naresh Grover, M.K.Soni Faculty of Engineering and Technology, Manav Rachna International University, Faridabad, India grovernr@rediffmail.com, dr_mksoni@hotmail.com Design of FPGA based 32-bit Floating Point Arithmetic Unit and verification of its VHDL code using MATLAB DOI: 10.5815/ijieeb.2014.01.01

[2] Yedukondala Rao Veeranki#1, R. Nakkeeran*2 , "Spartan 3E Synthesizable FPGA Based Floating-Point Arithmetic Unit" International Journal of Computer Trends and Technology (IJCTT) - volume4Issue4 –April 2013

[3] Murali Krishna Pavuluri1 and T.S.R. Krishna Prasad2 and Ch.Rambabu3DESIGN AND IMPLEMENTATION OF COMPLEX FLOATING POINT PROCESSOR USING FPGA,International Journal of VLSI design & Communication Systems (VLSICS) Vol.4, No.5, October 2013

[4] L.F. Rahman, Md. Mamun, M.S. Amin, "VHDL Environment for Pipeline Floating Point Arithmetic Logic Unit Design and Simulation"Department of Electrical, Electronic and Systems Engineering, Universiti Kebangsaan Malaysia, 43600 Bangi,Selangor, Malaysia Journal of Applied Sciences Research, 8(1): 611-619, 2012

[5] Pardeep Sharma Ajay Pal Sing,Implementation of Floating Point Multiplier on Reconfigurable Hardware and Study its Effect on 4 input LUT's h Volume 2, Issue 7, July 2012

[6] Shuchita Pare*1 Dr. Rita Jain*2 "32 Bit Floating Point Arithmetic Logic Unit ALU Design and Simulation" international journal of imerging trend in electronics and computer science volume 1 issue, 2012

[7] Prashanth B.u.v \ P.Anil Kumai, .G Sreenivasulu3 iMaintenance Engineer, DST-PURSE, 2M. TECH Student , 3Associate Professor,S. V University, Tirupati-517502 "Design & Implementation of Floating point ALU on a FPGA Processor" International Conference on Computing, Electronics and Electrical Technologies [ICCEET],2012

[8] Dhiraj Sangwan1& Mahesh K. Yadav2, Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic, International Journal of Electronics Engineering, 2(1), 2010, pp. 197-203

[9] Mamu Bin Ibne Reaz, MEEE, Md. Shabiul Islam, MEEE, Mohd. S. Sulaiman, MEEEFaculty of Engineering, Multimedia University, 63 100 Cybejaya, Selangor, Malaysia "Pipeline Floating Point ALU Design using VHDL" ICSE2002 Proc. 2002

[10] Shabiul Islam, MEEE, Mohd. S. Sulaiman, Mamu Bin Ibne Reaz MEEEFaculty of Engineering, Multimedia University, 63 100 Cybejaya, Selangor, Malaysia ICSE2002 Proc. 2002, Penang, Malaysia Pipeline Floating Point ALU Design using VHDL, MEEE, Md. 2002

[11] 1Rajit Ram Singh 2Vinay Kumar Singh 3poornima shrivastav 4Dr. GS Tomar "VHDL environment for floating point Arithmetic Logic Unit – ALU design and simulation"